2012 International Conference on Innovation and Information Management (ICIIM 2012)
IPCSIT vol. 36 (2012) © (2012) IACSIT Press, Singapore

BDB FTL: Design and Implementation of a Buffer-Detector Based Flash Translation Layer

Hong Zhou[1, a+], Jiang Jiang[1,b] , Ting Liu[1,c] and Xing Han[1,d]

[1] School of Microelectronic in Shanghai Jiaotong University, China

Abstract. The quickly development of flash memory technologies has consolidated the leadership of flash memory as a secondary storage solution. However the poor performance of random writes has been a bottleneck of flash memory. In this paper, we propose a creative Flash Translation Layer named BDB FTL to more exactly detect the sequential locality of data and to perform random writes in parallel through different available channels. Experiment results shown excellent performance of our scheme. A random-write dominant I/O trace from an OLTP application shows a 20.3% improvement in average write response time compared with the state-of-the-art FTL scheme DFTL.

Keywords: Solid-State Drives; FTL; Random Writes; Parallel Channels; Write Buffer; Response Time

1. Introduction

Nowadays, Flash memory has been widely used as a storage device both for embedded systems and general-purpose computing markets. However, it is impossible to use flash memory in a straightforward way due to several limitations: erase before write(unable of overwrite directly), low speed of erase and limited erase cycles. To solve these problems, the Flash Translation Layer (FTL), which performs the logical-to-physical address translations and realizes out of place updates, has been developed.

FTL schemes can be categorized into page level, block level and hybrid mapping schemes. Page level mapping has high block utilization and good performance for both reads and writes. Though page level is very flexible, it requires large amount of memory to store address mapping information, which is unacceptable. On the other hand, block level requires less memory and is good for read-intensive data, but it is bad for write-intensive data, which causes more erase operations. Hybrid FTL schemes such as BAST[1], FAST[2] and SuperBlock FTL[3] combine the characteristics of both page level and block level, making an effort to "update" block utilization and to reduce the garbage collection overheads.

It is a trade-off between mapping granularity and memory space requirements. DFTL[4] proposed an efficient way to selectively cache page-level address mappings, which has solved the essential problem of various of FTLs mentioned above.

2. Background and Related work

Existing Flash SSDs exhibit good sequential/random read performance but have a bottleneck of random write performance[5,10,11,12,13,14,15], due to the expensive full merge caused by random writes. Full merge is a kind of merge operation during Garbage Collection, which can cause more extra read and write operation.

+ Corresponding author
E-mail address: [a]Microe.zhouhong@gmail.com, [b]jiangjiang@ic.sjtu.edu.cn,
[c]liuting@ic.sjtu.edu.cn, [d]xinghan@ic.sjtu.edu.cn

LAST indicates[6] that in general-purpose computing systems the inefficiency of the existing FTLs can be deteriorated, due to different write access patterns. For example, most write requests in a MP3 player and movies are sequential, but random write is dominant in OLTP applications. On the other hand, the write request pattern is more complex in general-purpose applications. To improve the performance of random writes, it is necessary to detect the locality of access data first.

LAST determines the locality type by comparing the size of each request with a threshold value. DFTL indicates[4] that the locality detector proposed by LAST[6] cannot correctly identify sequential writes when the small-sized write has sequential locality.

Contributions: This paper mainly makes the following three contributions:

First, to significantly reduce the amount of memory used to store address mapping information, BDB FTL adopts selective cache mapping table (CMT) proposed by DFTL. Based on DFTL, we propose a new method called Buffer-Detector Based FTL(BDB FTL) mainly for two purposes: first, to detect the locality type more correctly which can overcome the inherent shortcoming of LAST; second, to store data of random locality type and sequential locality type separately to apply different write styles for different locality type.

Second, to exploit both the temporal locality and sequential locality, BDB FTL scheme employs different policies to get the new physical address for data accessing according to the locality type judged by Buffer-Detector.

Third, we propose a creative way for random writes named "Selective Parallel Channels Write Policy", which can efficiently enhance the performance of random writes.

3. Overall System Architecture of BDB FTL

The system architecture of BDB FTL is shown in Figure 1: Based on the architecture of DFTL, BDB FTL adds the component of Buffer-Detector. Buffer-Detector consists of: Sequential-Buffer, Random-Buffer and Address-Subtracter. Sequential-Buffer and Random-Buffer are a set of registers, storing data of sequential locality and random locality respectively, as shown in Figure 1. And Address-Subtracter composed of Pre-Register and Current-Register. Abbreviation of GC and WL stands for Garbage Collector and Wear Lever respectively. The package module represents a group of flash dies that share a bus channel.

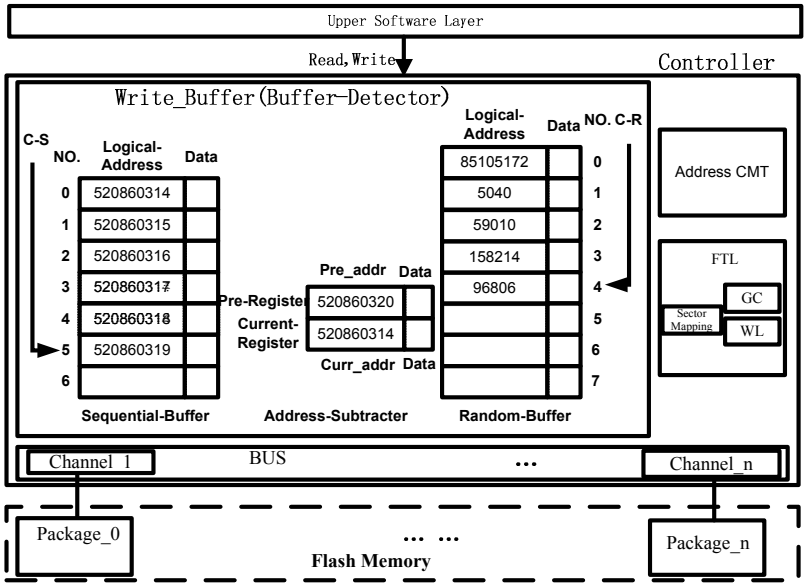

Fig. 1: Overall System Architecture Diagram of BDB FTL

If the request is writing, for each coming page, the data will be buffered into the write buffer with its address, waiting to judge the locality type of the coming address. To invalidate the old physical page, BDB FTL will firstly get the mapping table from Cache Mapping Table (CMT) stored in the RAM. If missed, BDB FTL will access Global Translation Directory (GTD) stored in Flash. According to the judgment of locality type, BDB FTL employs different write styles to get an empty page to write into.

On the other hand, if the request is read, BBD FTL will firstly judge whether the data to read is in the write buffer. If it is, BBD FTL reads directly from Buffer-Detector otherwise it is inevitably to get the mapping information from CMT or GTD. And then BDB FTL reads the correct data from Flash.

4. Design and Process Flow of BDB FTL

In our write scheme, every new coming page will be stored in Current-Register temporarily. And Pre-Register holds the previous coming page. Address-Subtracter performs the operation of subtraction between the address of current page and previous page. If the result of the operation of Address-Subtracter is 1, the previous page will be copied into Sequential-Buffer until Sequential-Buffer is full. Otherwise, previous page will be stored into Random-Buffer until the number of records in "Random-Buffer" reaches the number of valid channels at present.

```
Store new logical address into Curr_Register ;
if(Buffer-Detector is in Sequential Mode){
    if(address of the current data minus address of the previous data equals 1){
        Flash memory ← Pre_Register; /*in the way of sequential writes*/
    else{
        Flash memory ← Pre_Register; /*the last data of sequential writes*/
        Switch Buffer-Detector to Random Mode;}}
 else if(Buffer-Detector is in Judge Mode){ /*Sequential-Buffer is neither empty nor full*/
    if(address of the current data minus address of the previous data equals 1){
        Sequential-Buffer ← Pre_Register;
        if(Sequential-Buffer is full){
            Write content of Sequential-Buffer to Flash memory and Switch Buffer-Detector to
Sequential Mode;}}
    else
        Switch Buffer-Detector to Random Mode;
        Do Selective Parallel Channels Write Policy; // in the way of random writes
 else if(Buffer-Detector is in Random Mode){
        if(address of the current data minus address of the previous data equals 1){
            Switch Buffer-Detector to Judge Mode and Write the content of Pre_Register to
Sequential-Buffer;}
        else
            Write the content of Pre_Register to Random-Buffer;
            if(the records of Random-Buffer is no less than that of valid channels at present){
                Do Selective Parallel Channels Write Policy; /*in the way of random
writes*/}}
```

Fig.2 Process Flow of Buffer-Detector.

The whole design flow of Buffer-Detector is shown in Figure 2. Note that "Selective Parallel Channels Write Policy" included in Figure 2 is shown in Figure 6. Section 5 gives a detail explanation for the way of "sequential writes" and "random writes".

5. Get New Physical Address of an Empty Page

When Buffer-Detector is in Sequential Mode, BDB FTL gets an empty page sequentially inside a block for "Sequential Writes". As figure 3 depicts, Addr_Pointer_Sequential keeps the record of the previous physical address for sequential writes. If no empty page can be found in the current block, another free block will be chose to write to.

Figure 4 shows how to get an empty page inside package i through the corresponding channel. Addr_Pointer_Random[i] keeps the record of the previous physical address for random writes in channel i, while Addr_Pointer_Random_Temp[i] is designed to find another physical address for the current random write. Note that if no empty page can be found in the current block, an empty page will be found inside another block no matter whether the block is an erased one or not, which is different from the case of "Sequential Writes".

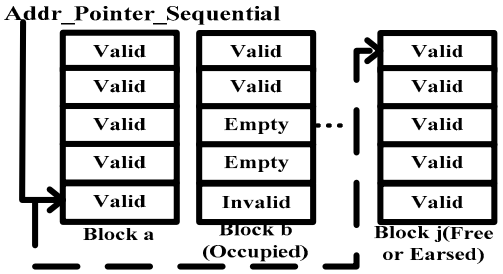

Fig.3: Find a free block for sequential writes

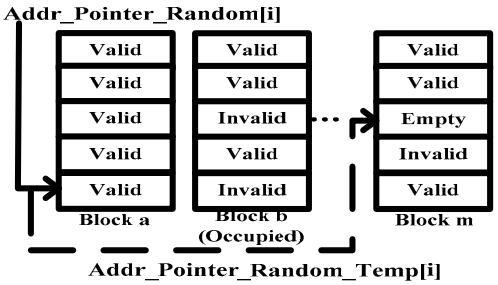

Fig.4: Find an empty page for random writes

6. Selective Parallel Channels Write Policy

Based on the correct judgment of the locality type of write data, a creative write policy called "Selective Parallel Channels Write Policy" is proposed by BDB FTL, which can greatly improve random write performance.

In "Random Writes", data will be written into different valid channels in parallel as shown in Figure 5. The status of each channel depends on write access patterns, Garbage Collection policy and Wear status. If the status of a channel is Invalid, it means the channel is suspended until the status is updated. The whole flow of the design of "Random Writes" is shown in Figure 6.

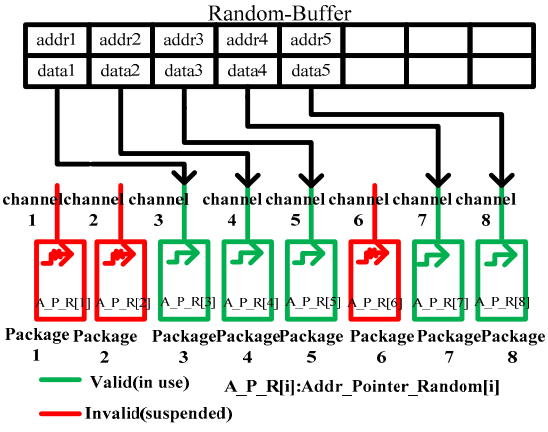

Fig. 5: Random Writes Policy

```
for(int i = 0; i<records; i++){ /* find an available channel for
each page and guarantee the success of write process*/
    if(the last channel has been checked){
        if(no channel is available) {/* no empty page can be found
from the whole SSD except preserved free blocks for Garbage
Collection */
            BDB FTL performs Garbage Collection Algorithm;
            Update the status for each channel after GC;
            Check from the first channel;}
    if((the status of current channel is valid)
            &&(write process is successful))
        finish the current loop and write the next record to another channel;
    else{ /* no empty page can be found from the current package*/
            suspend the current channel;
    set the channel status as "Invalid";}}
```

Fig. 6: Selective Parallel Channels Write Policy

7. Simulation and Analysis

Our simulator is based on an event-driven simulator that follows the object-oriented programming paradigm for modularity named FlashSim[7] developed by Pennsylvania State University with some modification.

Table1shows a part of parameters in our simulation, and some traces are collected from activities of desktop PC running applications such as mp3, movie and general-purpose applications by a Diskmon tool developed by Microsoft Corp[8]. The other trace is from an OLTP application running at a financial institution[9], which is random writes dominant.

Table 1: A part of Simulation Parameters

Ram Read Delay	Ram Write Delay	Bus Control Delay	Bus Data Delay	Max_Valid_ Channels	Reg Read Delay	Reg Write Delay	Buffer Size	Page_read_de lay	Page_write _delay	Block_erase _delay
0.01ms	0.01ms	0.02ms	0.01ms	8	0.001ms	0.001ms	7	0.025ms	0.2ms	1.5ms

Figure 7 demonstrates a 20.3% improvement in average write response time based on the trace of OLTP application in BDB FTL compared with DFTL. And as shown in Figure 8, our proposed scheme costs much less Full Merges. So BDB FTL causes less extra read and write operations during Garbage Collection compared with DFTL.

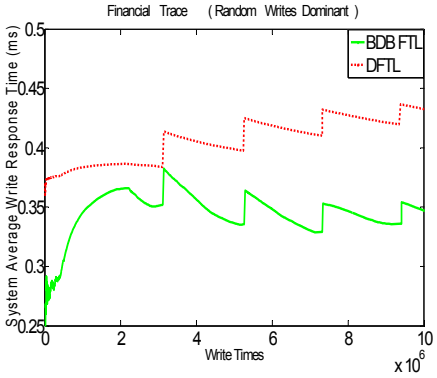

Fig. 7: Average Response Time of BDB FTL and DFTL based on Financial Trace.

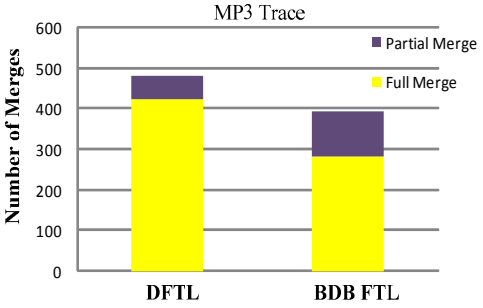

Fig. 8: Number of Full Merges and Partial Merges based on MP3 Trace.

8. Summary

In this paper, we propose a "Buffer-Detector" Based Flash Translation Layer, called BDB FTL. The design of BDB FTL is to detect the sequential locality of data more exactly and to perform "Random Writes" in parallel to take full use of available channels. Experiment results show a significant improvement in write performance of BDB FTL compared with DFTL. The cost of our scheme is that BDB FTL needs two sets of registers, which means more hardware resources and more power consumption.

9. References

[1] T. Chung, D. Park, S. Park, D. Lee, S. Lee, and H. Song. System Software for Flash Memory: A Survey. In *Proceedings of the International Conference on Embedded and Ubiquitous Computing*, ages 394–404, August 2006.

[2] S. Lee, D. Park, T. Chung, D. Lee, S. Park, and H. Song. A Log Buffer based Flash Translation Layer Using Fully Associative Sector Translation. *IEEE Transactions on Embedded Computing Systems*, 6(3):18, 2007. ISSN 1539–9087.

[3] J. Kang, H. Jo, J. Kim, and J. Lee. A Superblock-based Flash Translation Layer for NAND Flash Memory. In *Proceedings of the International Conference on Embedded Software (EMSOFT)*,pages 161–170, October 2006. ISBN 1-59593-542-8.

[4] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: a Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings, " in *ASPLOS*,2009.

[5] X.-Y. Hu and R. Haas, "The fundamental limit of flash random write performance: Understanding, analysis and performance modelling," IBM Research Report, *RZ 3771, Mar*. 2010.

[6] S. Lee, D. Shin, Y. Kim, and J. Kim. LAST: Locality-Aware Sector Translation for NAND Flash Memory-Based Storage Systems. In *Proceedings of the International Workshop on Storage and I/O Virtualization, Performance, Energy, Evaluation and Dependability (SPEED2008),* Feburary 2008.

[7] K. Youngjae, T. Brendan, G. Aayush, and U. Bhuvan. FlashSim: A Simulator for NAND Flash-based Solid-State Drives. In *The First International Conference on Advances in System Simulation,* 2009.

[8] Diskmon tool developed by Microsoft Corp. http://technet.microsoft.com/en-us/sysinternals/bb896646

[9] OLTP Trace from UMass Trace Repository. OLTP Trace from UMass Trace Repository. http://traces.cs.umass.edu/index.php/Storage/Storage

[10] D. Ajwani, I. Malinger, U. Meyer, and S. Toledo. Characterizing the performance of flash memory storage devices and its impact on algorithm design. In *Proceedings of the Workshop* in Experimental Algorithms, pages 208–219, 2008.

[11] A. Birrell, M. Isard, C. Thacker, and T. Wobber. A design for high-performance flash disks. *ACM SIGOPS Operating Systems Review,* 41:88–93, 2007.

[12] L. Bouganim, B. Jonsson, and P. Bonnet. uFLIP: Understanding flash IO patterns. In *Proceedings of the 4th Biennial Conference on Innovative Data Systems Research (CIDR),* Jan.2009.

[13] F. Chen, D. A. Koufaty, and X. Zhang. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *Proceedings of the ACM SIGMETRICS/Performance,* 2009.

[14] M. Polte, J. Simsa, and G. Gibson. Enabling enterprise solid state disks performance. In Proceedings of the First Workshopon Integrating Solid-state Memory into the Storage Hierarchy (WISH), held in conjunction with ASPLOS, 2009.

[15] R. Stoica, M. Athanassoulis, R. Johnson, and A. Ailamaki. Evaluating and repairing write performance on flash devices.In *Proceedings of the 5th International Workshop on Data Management on NewHardware,* 2009.